METEOR

目錄

Meteor 中文文档

> 来源：Meteor 中文文档

Meteor 是一个用于构建现代应用的超简单的开发环境。之前用最好的工具，都需要花费数周时间的事情，现在用***Meteor***，只需数小时。

Web最初被设计成和70年代大型机工作方式相同。服务器渲染完成一个页面并通过网络发送到终端。无论用户做了什么，服务端都会重新渲染整个页面。 这种模式在Web上持续了十多年。从而产生了LAMP，Rails，Django，PHP。

但是现在，一个非常牛的团队，他们有着充足的预算和长远的规划，使我们可以构建运行在客户端的javascript应用。这些应用拥有出色的界面。 无需刷新网页，而是实时响应：任何一个客户端产生变化都会立即反映到所有人的屏幕。

他们经过一番努力推出了Meteor。Meteor使构建现代应用变得简单而有趣。用一个周末的时间或是在黑客马拉松上，你就可以构建一个完整的应用。 你无需再准备服务器资源，或是部署API到云端，不用管理数据库或是纠缠于ORM层，不用再在javascript和Ruby之间来回切换，也不用再广播无效数据给客户端。

快速开始!

Meteor 支持 OS X, Windows, and Linux.

用 Windows? 下载官方安装包.

用 OS X 或是 Linux? 通过命令行安装Meteor官方最新版：

```
$ curl https://install.meteor.com/ | sh
```

Windows 安装包支持 Windows 7, Windows 8.1, Windows Server 2008, 和 Windows Server 2012。命令行安装，支持 Mac OS X 10.7 (Lion) 及其以上版本, Linux x86 和 x86_64 架构.

安装完成之后，创建一个项目：

```
$ meteor create myapp
```

在本地运行：

```
$ cd myapp
$ meteor
# Meteor server running on: http://localhost:3000/
```

然后，打开一个命令行窗口，发布到线上（我们提供的一个免费服务器）

```
$ meteor deploy myapp.meteor.com
```

Meteor的理念

- *Data on the Wire*. Meteor不在网络上发送HTML。服务器发送数据，让客户端来渲染它。
- *One Language.* Meteor使你可以用javascript来实现应用的客户端和服务端
- *Database Everywhere*. 在客户端和服务端，你可以使用相同的方法来使用数据库。
- *Latency Compensation*. 在客户端，Meteor 预加载数据以及本地模拟的模式使它看起来就像是服务端方法调用立即返回了一样。
- *Full Stack Reactivity*. 在Meteor中，默认是实时性的。所有层面，从数据库到模板，都会在需要的时候自动更新。
- *Embrace the Ecosystem*. Meteor是开源的，并且集成了现有的开源工具和框架。
- *Simplicity Equals Productivity*. 让事情看起来简单最好的方式就是让它真的成为简单地事。Meteor的主要功能点的API非常干净、经典漂亮。

学习资源

有很多社区资源可以帮助你开发应用。如果你对Meteor感兴趣，希望你能参与其中！

教程

快速开始Meteor 官方教程!

Stack Overflow

对于技术问题，提问、寻找答案最好的去处就是 Stack Overflow. 确保给你的问题添加 `meteor` 标签。

论坛

访问 Meteor discussion forums宣布项目，寻求帮助，讨论社区或是讨论核心模块的变动。

GitHub

核心代码在GitHub上. 如果你会写代码或是提issue，我们很想得到你的帮助。 对于如何开始，请先阅读Contributing to Meteor

命令行工具

meteor help

获取 meteor 命令行使用帮助。运行 meteor help 会列出meteor 所有命令。运行 meteor help <command> 会打印出关于 meteor <command> 的详细帮助。

meteor create <name>

创建一个名为 <name> 的子目录，并在里面新建一个Meteor应用。

meteor run

使用Meteor本地开发服务器运行当前应用，地址为：http://localhost:3000

meteor debug

附带着Node Inspector 运行项目，这样你就可以一步一步跟踪服务端代码。更多信息查看 [meteor debug]

meteor deploy <site>

打包你的应用，并发布到 <site> 。如果你发布到 <your app>.meteor.com ，Meteor提供免费的主机，只要 <your app> 的名字还未被其他人使用。

meteor update

升级Meteor到最新发布版，然后（如果 meteor update 是在一个应用目录中执行的）升级当前项目使用的包到最新兼容版。

meteor add

添加一个包（或多个）到Meteor项目中。要查询可用的包，使用 meteor search 命令。

meteor remove

移除之前添加到项目中的包。查看项目使用的包列表，用 meteor list 命令。

meteor mongo

打开一个MongoDB shell来查看、操作数据库中的集合。注意：你必须先运行当前应用(在另外的命令行)， meteor mongo 才能连接到应用的数据库

meteor reset

重置当前项目为初始状态。移除所有本地数据。

如果你经常使用 `meteor reset` ，但是又不想丢失一些初始数据，考虑使用 `Meteor.startup` 在服务第一次启动时重建这些数据。

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    if (Rooms.find().count() === 0) {
      Rooms.insert({name: "Initial room"});
    }
  });
}
```

文件结构

对于该如何组织应用的文件结构，Meteor是非常灵活的。它会自动加载所有文件，所以不需要再用 `<script>` 或 `<link>` 标签来引入javascript和CSS。

文件的默认加载

如果某个文件在下面提到的特殊文件夹之外，Meteor会做如下处理：

1. HTML模板编译完成后发送到客户端。详细信息参见the templates section。
2. CSS文件发送到客户端。在生产模式下，会自动合并、压缩。
3. Javascript被加载到客户端和服务端。可以使用 `Meteor.isClient` 和 `Meteor.isServer` 来控制特定代码的运行位置。

如果你想更好的控制哪些javascript代码加载到客户端或是服务端，可以使用下面列出的特殊文件夹。

特殊文件夹

`/client`

`/client` 文件夹中所有文件都只发送到客户端。用来放置HTML，CSS和UI相关的javascript代码。

`/server`

`/server` 文件夹中所有文件都只提供给服务端使用，不会发送到客户端。用来放置不应该被客户端看到的敏感逻辑和数据。

`/public`

`/public` 文件夹中的文件会原样发送到客户端。用来放置资源，例如：图片。假设有张图片 `/public/background.png` ，在HTML中用 `<img src='/background.png'/>` 引用或是在CSS中用 `background-image: url(/background.png)` 引用。 注意图片URL中不包含 `/public` 。

`/private`

`/private` 文件夹中的文件只能由服务端代码通过 `Assets` API 来获取，客户端无法获取。

更多关于文件加载顺序和特殊文件夹的说明参见Structuring Your App section

开发手机应用

当你用Meteor开发完成web app之后，只需几个命令，就可以轻松地构建一个native包装，并发布到Google Play Store或是iOS App Store。 我们用了很大精力，使相同的包和API在桌面端和移动端都能工作，所以你无须担心大量的移动应用开发相关的边界情况。

安装手机 SDKs

安装Android 或ios 开发工具：

```
meteor install-sdk android     # for Android
meteor install-sdk ios         # for iOS
```

添加平台

添加相关平台到你的应用：

```
meteor add-platform android    # for Android
meteor add-platform ios        # for iOS
```

运行模拟器

```
meteor run android             # for Android
meteor run ios                 # for iOS
```

在设备上运行

```
meteor run android-device      # for Android
meteor run ios-device          # for iOS
```

配置APP图标和metadata

可以用特殊的文件`mobile-config.js`配置APP的图标、标题、版本号、启动画面以及其他metadata。

更多关于Meteor对移动端的支持参见GitHub wiki page。

The Meteor API

Javascript代码可以运行在两种环境：客户端(浏览器)，和服务端(服务器上的Node.js容器)。在API 参考中，我们会指出每个函数是否可以在客户端，或是服务端，或是*Anywhere(*客户端和服务端*)*被调用。

模板

在Meteor中，视图定义在 模板 。模板就是包含动态数据的HTML代码段。你可以通过javascript给模板插入数据，或是监听模板事件。

用HTML定义模板

模板定义在 `.html` 文件中，可以放在项目任何位置，除了 `server` , `public` , `private` 文件夹

每个 `.html` 文件可以包含任意数量的顶级元素： `<head>` , `<body>` 或是 `<template>` 。 `<head>` , `<body>` 标签里的代码会附加到HTML页面中对应的标签， `<tempalte>` 标签里的代码可以用 `{{> templateName}}` 引入，如下面的例子所示。模板可以引入多次—模板的主要目的之一就是避免重复手写相同的HTML。

```
<!-- add code to the <head> of the page -->
<head>
  <title>My website!</title>
</head>

<!-- add code to the <body> of the page -->
<body>
  <h1>Hello!</h1>
  {{> welcomePage}}
</body>

<!-- define a template called welcomePage -->
<template name="welcomePage">
  <p>Welcome to my website!</p>
</template>
```

`{{ ... }}` 是Spacebars语法，Meteor使用Spacebars给HTML增加功能。如上所示，利用它你可以引入模板。 使用Spacebars，你可以显示从 *helpers* 中获取的数据。Helpers 用javascript来写，既可以是简单值，也可以是函数。

Template.myTemplate.helpers(helpers)

Client

Specify template helpers available to this template.

Arguments

helpers Object

Dictionary of helper functions by name.

给 `nametag` 模板定义一个叫做 `name` 的helper(在javascript里)：

```
Template.nametag.helpers({
  name: "Ben Bitdiddle" });
```

`nametag` 模板本身 (在HTML里):

```
<!-- In an HTML file, display the value of the helper -->
<template name="nametag">
  <p>My name is {{name}}.</p>
</template>
```

Spacebars还有几个方便使用的控制结构，可以使视图更加动态：

- `{{#each data}} ... {{/each}}` - 循环 `data` 里的每一项，每一项都会显示一次 `#each` 块里的HTML。
- `{{#if data}} ... {{else}} ... {{/if}}` - 如果 `data` 为 `true` ，显示第一个块，反之，显示第二个块。
- `{{#with data}} ... {{/with}}` - 设置内部HTML的数据上下文，然后显示。

每一个嵌套的 `#each` 或 `#with` 块都有自己的 数据上下文 ,数据上下文是一个对象，它的属性在块内部可以用作helper。例如： `#with` 块，它的数据上下文就是跟在它后面， `}}` 之前的变量值。 `#each` 块，循环到的元素会作为当前数据上下文。

例如， `people` helper 的值为：

```
Template.welcomePage.helpers({
  people: [{name: "Bob"}, {name: "Frank"}, {name: "Alice"}]
});
```

然后你可以用一个 `<p>` 标签列表显示每一个人的姓名：

```
{{#each people}}
  <p>{{name}}</p>
{{/each}}
```

或是用上面的"nametag"模板代替 `<p>` 标签：

```
{{#each people}}
  {{> nametag}}
{{/each}}
```

记住：helper可以是简答值，也可以是函数。例如，要显示登录用户的用户名，你可以定义一个叫做 `username` 的helper：

```
// in your JS file
Template.profilePage.helpers({
  username: function () {
    return Meteor.user() && Meteor.user().username;
  }
});
```

现在，每次使用 `username` helper的时候，都会调用上面的helper函数来确定用户名：

```
<!-- in your HTML -->
<template name="profilePage">
  <p>Profile page for {{username}}</p>
</template>
```

Helper可以接收参数。例如，

```
Template.post.helpers({
  commentCount: function (numComments) {
    if (numComments === 1) {
      return "1 comment";
    } else {
      return numComments + " comments";
    }
  }
});
```

参数放在大括号里，helper名之后：

```
<p>There are {{commentCount 3}}.</p>
```

上面定义的helper都关联到特定的模板，可以用 `Template.registerHelper` 定义所有模板都能用的helper。

关于Spacebars更详细的文档参见README on GitHub。后面的 `Session` , `Tracker` , `Collections` , 和 `Accounts` 小节会有如何给模板添加动态数据的讨论。

Template.myTemplate.events(eventMap)

Client

Specify event handlers for this template.

Arguments

eventMap Event Map

Event handlers to associate with this template.

传给 `Template.myTemplate.events` 的事件map,用事件描述符作为key,事件处理函数作为value。事件处理函数接收两个参数：事件对象和模板实例。事件处理函数中通过 `this` 获取数据上下文。

假设有下面的模板：

```
<template name="example">
  {{#with myHelper}}
    <button class="my-button">My button</button>
    <form>
      <input type="text" name="myInput" />
      <input type="submit" value="Submit Form" />
    </form>
  {{/with}}
</template>
```

调用 `Template.example.events` 给模板增加事件处理器：

```
Template.example.events({
  "click .my-button": function (event, template) {
    alert("My button was clicked!");
  },
  "submit form": function (event, template) {
    var inputValue = event.target.myInput.value;
    var helperValue = this;
    alert(inputValue, helperValue);
  }
});
```

key的前半部分是要捕获的事件名称。几乎支持所有的DOM事件，常见的有：`click`，`mousedown`，`mouseup`，`mouseenter`，`mouseleave`，`keydown`，`keyup`，`keypress`，`focus`，`blur`，和 `change`。

key的后半部分是CSS选择器，用来指明要监听的元素。几乎支持所有JQuery支持的选择器.

无论何时，选定元素上触发了监听的事件时，对应的事件处理函数就会被调用，参数为：DOM 事件对象和模板实例。详细信息参见Event Maps section

Template.myTemplate.onRendered

Client

Register a function to be called when an instance of this template is inserted into the DOM.

Arguments

callback Function

A function to be added as a callback.

用这个方法注册的函数会在*Template.myTemplate*模板的每个实例第一次插入到页面的时候调用一次。

这个回调函数可以用来集成那些不适应Meteor自动视图渲染机制,并且需要在每次HTML插入到页面时进行初始化的第三方库。 你可以在 `onCreated` 和 `onDestroyed` 回调中执行对象初始化或是清理工作。

例如，要使用HighlightJS库高亮 `codeSample` 模板中所有 `<pre>` 元素，你可以传递如下回调函数给 `Template.codeSample.onRendered` ：

```
Template.codeSample.onRendered(function () {
  hljs.highlightBlock(this.findAll('pre'));
});
```

在回调函数中， `this` 指向一个template instance对象实例，that is unique to this inclusion of the template and remains across re-renderings.可以使用方法 `this.find` 和 `this.findAll` 来获取模板渲染后的HTML DOM节点。

Template instances

一个模板实例对象代表文档对模板的一次引入。模板实例可以用来获取模板中的HTML元素，还可以给模板实例附加属性，属性会在模板响应式更新中保持，不会丢失。

在好几个地方都可以获取到模板实例对象：

1. 在 `created` , `rendered` 和 `destroyed` 模板回调中，this指向模板实例
2. 事件处理器的第二个参数
3. 在Helper中，通过 `Template.instance()` 获取模板实例

你可以选择给模板实例附加属性，来跟踪模板相关的状态。例如，当使用Google Maps API时，你可以附加 `map` 对象到当前模板实例，这样就可以在Helper和事件处理器中引用 `map` 对象。用 `onCreated` 和 `onDestroyed` 回调函数来执行初始化或清理工作。

template.findAll(selector)

Client

Find all elements matching `selector` in this template instance.

Arguments

selector String

The CSS selector to match, scoped to the template contents.

`template.findAll` 返回一个符合 `selector` 的DOM元素数组。也可以使用 `template.$` ，它的工作方式和JQuery的 `$` 函数一样，但是只返回 `template` 内部的元素。

template.find(selector)

Client

Find one element matching `selector` in this template instance.

Arguments

selector String

The CSS selector to match, scoped to the template contents.

`find` 类似 `findAll` 但是只返回找到的第一个元素。和 `findAll` 一样， `find` 只返回模板内部的元素。

Session

在客户端， `Session` 提供了一个全局对象，可以用它来保存任意的键值对。例如：保存列表中当前选中项。

`Session` 的特殊之处在于，它是响应式的。如果在template helper或 `Tracker.autorun` 里调用了 `Session.get("myKey")` ，那么无论何时调用 `Session.set("myKey", newValue)` 都会触发相应的模板片段自动重新渲染。

Session.set(key, value)

Client

将session里键key的值设为value。并通知所有监听器：键key的值发生了变化（例如：通知那些调用 `Session.get` 获取键 `key` 值的模板helper或是autorun，重新渲染模板或是重新执行 `Tracker.autorun` ）。

Arguments

key String

The key to set, eg, `selectedItem`

value EJSON-able Object or undefined

The new value for `key`

Session.get(key)

Client

获取session里键key的值。如果是在一个响应式计算*(reactive computation)*内部。当通过 `Session.set` 修改键key的值时，会作废对应的计算。这个方法会返回键key值的克隆副本，所以如果键key的值是一个对象或数组，修改返回值并不会改变session里键key的值。

Arguments

key String

The name of the session variable to return

例如：

```
<!-- In your template -->
<template name="main">
  <p>We've always been at war with {{theEnemy}}.</p>
</template>
```

```
// In your JavaScript
Template.main.helpers({
  theEnemy: function () {
    return Session.get("enemy");
  }
});

Session.set("enemy", "Eastasia");
// Page will say "We've always been at war with Eastasia"

Session.set("enemy", "Eurasia");
// Page will change to say "We've always been at war with Eurasia"
```

`Seesion` 带我们初次体会到了响应式的魅力，视图会在必要时自动更新，无需手动调用 `render` 函数。 在下一节，将会学习如何使用Tracker，一个非常轻量的库，使响应式成为可能。

Tracker

Meteor 包含一个简单地依赖跟踪系统，使用它可以在 `Session` 变量、数据库查询或其它数据源发生变化时自动重新渲染模板或是重新运行某些函数。

和其它的依赖跟踪系统不同，不需要手工声明依赖— 它就能工作。它的机制简单而高效。一旦你用 `Tracker.autorun` 初始化了一个计算(computation), `Tracker` 自动记录使用到的数据。当这些数据发生变化时，计算就会自动重新运行。这就是为什么当helper functions返回新数据时模板知道如何重新渲染。

Tracker.autorun(runFunc, [options])

Client

Run a function now and rerun it later whenever its dependencies change. Returns a Computation object that can be used to stop or observe the rerunning.

Arguments

runFunc Function

The function to run. It receives one argument: the Computation object that will be returned.

Options

onError Function

Optional. The function to run when an error happens in the Computation. The only argument it recieves is the Error thrown. Defaults to the error being logged to the console.

`Tracker.autorun` 使你可以声明一个依赖响应式数据源的函数，无论何时数据源发生变化，函数都会被重新执行。

例如，可以监测一个 `Session` 变量，设置另外一个：

```
Tracker.autorun(function () {
  var celsius = Session.get("celsius");
  Session.set("fahrenheit", celsius * 9/5 + 32);
});
```

或者可以等待session变量成为一个特定值，执行一些特定操作。如果想阻止回调函数进一步重新运行，可以调用计算(computation)对象的 `stop` ，计算对象会作为回调函数的第一个参数传入：

```
// Initialize a session variable called "counter" to 0
Session.set("counter", 0);

// The autorun function runs but does not alert (counter: 0)
Tracker.autorun(function (computation) {
  if (Session.get("counter") === 2) {
    computation.stop();
    alert("counter reached two");
  }
});

// The autorun function runs but does not alert (counter: 1)
Session.set("counter", Session.get("counter") + 1);

// The autorun function runs and alerts "counter reached two"
Session.set("counter", Session.get("counter") + 1);

// The autorun function no longer runs (counter: 3)
Session.set("counter", Session.get("counter") + 1);
```

`Tracker.autorun` 第一次被调用的时候，回调函数立即被执行，如果此时 `counter === 2` ，那么就会alert，然后立即停止。在上面的例子中，当 `Tracker.autorun` 被调用时， `Session.get("counter") === 0` ，所以第一次什么都不会发生，每次 `counter` 发生变化时，回调函数都会重新运行，直到当 `counter` 等于 `2` 的时候， `computation.stop()` 被调用。

如果autorun在第一次执行时抛出了异常，那么计算(computation)会自动停止，以后也不会重新运行。

关于 `Tracker` 的工作原理和高级用法，参见Meteor 手册里的Tracker一节，里面有更加详细的说明。

Collections

Meteor用集合保存数据。集合里保存的Javascript对象叫做文档。使用 `new Mongo.Collection` 声明一个集合。

new Mongo.Collection(name, [options])

Anywhere

Constructor for a Collection

Arguments

name String

The name of the collection. If null, creates an unmanaged (unsynchronized) local collection.

调用 `Mongo.Collection` 构造函数创建一个集合对象，它的行为就像MongoDB 的集合。如果在创建集合时传入了一个name参数，那么声明的就是一个持久性集合 — 保存在服务端并可以发布到客户端。

要想使客户端代码和服务端代码都可以通过相同的API访问相同的集合，最好在一个客户端和服务端都能加载到的javascript文件中把集合声明为全局变量。

示例：声明两个命名的，持久性的集合作为全局变量：

```
// In a JS file that's loaded on the client and the server
Posts = new Mongo.Collection("posts");
Comments = new Mongo.Collection("comments");
```

如果传入 `null` 作为name参数，那么创建出来的就是一个本地集合。本地集合不会在客户端和服务端之间进行同步；它只是javascript对象的临时集合，支持Mongo-style `find` , `insert` , `update` , 和 `remove` 操作。

默认情况下，Meteor会自动发布所有集合里的文档到每一个连接上的客户端。要禁用此行为，必须移除 `autopublish` 包：

```
$ meteor remove autopublish
```

然后，使用 `Meteor.publish` 和 `Meteor.subscribe` 来指定集合的哪一部分发送到哪些客户端。

使用 `findOne` 和 `find` 从集合里检索文档。

collection.findOne([selector], [options])

Anywhere

Finds the first document that matches the selector, as ordered by sort and skip options.

Arguments

selector Mongo Selector, Object ID, or String

A query describing the documents to find

Options

sort Mongo Sort Specifier

Sort order (default: natural order)

skip Number

Number of results to skip at the beginning

fields Mongo Field Specifier

Dictionary of fields to return or exclude.

`findOne` 方法可以从集合里查找特定的文档。调用 `findOne` 时，通常都会传入一个特定文档的 `_id`：

```
var post = Posts.findOne(postId);
```

然而，也可以给 `findOne` 传入一个Mongo 选择器，Mongo选择器是一个对象，指明了目标文档要满足的一系列属性。 例如，下面的选择器

```
var post = Posts.findOne({
  createdBy: "12345",
  title: {$regex: /first/}
});
```

会匹配到下面的文档

```
{
  createdBy: "12345",
  title: "My first post!",
  content: "Today was a good day." }
```

关于MongoDB query operators 例如 `$regex` , `$lt` (小于), `$text` (文本搜索)的更多信息参见MongoDB documentation。

一个非常有用但是不那么明显的功能就是Mongo 选择器可以匹配数组里的元素。例如：下面的选择器

```
Post.findOne({
  tags: "meteor" });
```

会匹配到下面的文档

```
{
  title: "I love Meteor",
  createdBy: "242135223",
  tags: ["meteor", "javascript", "fun"] }
```

`findOne` 方法和 `Session.get` 一样，也是响应式的，也就是说，如果你在template helper或是 `Tracker.autorun` 回调函数中使用了 `findOne` ，如果 `findOne` 返回的文档发生了变化，那么模板就会自动重新渲染，计算会重新执行。

注意，如果 `findOne` 没有找到任何文档，则返回 `null` ，通常发生在文档还没加载或是已经从集合中移除，所以要有处理 `null` 值的准备。

collection.find([selector], [options])

Anywhere

Find the documents in a collection that match the selector.

Arguments

selector Mongo Selector, Object ID, or String

A query describing the documents to find

Options

sort Mongo Sort Specifier

Sort order (default: natural order)

skip Number

Number of results to skip at the beginning

limit Number

Maximum number of results to return

fields Mongo Field Specifier

Dictionary of fields to return or exclude.

`find` 方法和 `findOne` 类似，不同的是，它不返回单一文档，而是返回一个MongoDB 游标。游标是一个特殊的对象，代表一个查询里会被返回的文档列表。可以在模板Helper里返回游标，或是其它可以返回数组的地方：

```
Template.blog.helpers({
  posts: function () {
    // this helper returns a cursor of
    // all of the posts in the collection
    return Posts.find();
  }
});
```

```
<!-- a template that renders multiple posts -->
<template name="blog">
  {{#each posts}}
    <h1>{{title}}</h1>
    <p>{{content}}</p>
  {{/each}}
</template>
```

要想从一个游标里检索当前的文档列表时，调用游标的 `.fetch()` 方法：

```
// get an array of posts
var postsArray = Posts.find().fetch();
```

记住，虽然调用 `fetch` 的计算会在数据变化时重新运行，但是the resulting array will not be reactive if it is passed somewhere else.

通过调用 `insert` , `update` , 或 `remove` 来修改保存在 `Mongo.Collection` 里的数据。

collection.insert(doc, [callback])

Anywhere

Insert a document in the collection. Returns its unique _id.

Arguments

doc Object

The document to insert. May not yet have an _id attribute, in which case Meteor will generate one for you.

callback Function

Optional. If present, called with an error object as the first argument and, if no error, the _id as the second.

下面的例子展示了如何插入文档到集合里：

```
Posts.insert({
  createdBy: Meteor.userId(),
  createdAt: new Date(),
  title: "My first post!",
  content: "Today was a good day." });
```

每个 `Mongo.Collection` 里的每个文档都有一个 `_id` 字段。它必须是唯一的，如果你没有提供，会自动生成。 `collection.findOne` 使用 `_id` 可以用来检索特定的文档。

collection.update(selector, modifier, [options], [callback])

Anywhere

Modify one or more documents in the collection. Returns the number of affected documents.

Arguments

selector Mongo Selector, Object ID, or String

Specifies which documents to modify

modifier Mongo Modifier

Specifies how to modify the documents

callback Function

Optional. If present, called with an error object as the first argument and, if no error, the number of affected documents as the second.

Options

multi Boolean

True to modify all matching documents; false to only modify one of the matching documents (the default).

upsert Boolean

True to insert a document if no matching documents are found.

这里的选择器和你传给 `find` 方法的是一致的，它可以匹配多个文档。修改器是一个对象，它指明了对匹配到的文档要做的修改。注意：除非你使用了 `$set` 操作符，否则 `update` 方法会直接用修改器替换整个匹配到的文档。

下面的例子展示了，设置所有标题包含"first"的文章的内容字段

```
Posts.update({
  title: {$regex: /first/}
}, {
  $set: {content: "Tomorrow will be a great day."}
});
```

关于支持的所有operators参见MongoDB documentation。

有一点需要注意：当在客户端调用 `update` 的时候，只能通过 `_id` 来查找文档。要使用所有可用的选择器，必须在服务端代码或是 method中调用 `update` 。

collection.remove(selector, [callback])

Anywhere

Remove documents from the collection

Arguments

selector Mongo Selector, Object ID, or String

Specifies which documents to remove

callback Function

Optional. If present, called with an error object as its argument.

`remove` 方法使用和 `find` 、 `update` 一样的选择器，并且从数据库中移除所有匹配到的文档。请小心使用 `remove` — 删掉的数据没办法恢复。

和 `update` 一样，客户端代码只能通过 `_id` 移除文档，而服务端代码和methods可以使用任何选择器移除文档。

collection.allow(options)

Server

Allow users to write directly to this collection from client code, subject to limitations you define.

Options

insert, update, remove Function

Functions that look at a proposed modification to the database and return true if it should be allowed.

在新创建的APP中，Meteor允许任何客户端和服务端代码调用 `insert` , `update` , 和 `remove` 。这是因为用 `meteor create` 创建的APP默认包含了 `insecure` 包，目的是简化开发。很显然，如果任何用户都可以修改数据库，这是很不安全的，所以移除 `insecure` 包，并声明一些权限规则是很重要的：

```
$ meteor remove insecure
```

一旦你移除了 `insecure` 包，就可以用 `allow` 和 `deny` 来控制谁可以执行哪些数据库操作。默认情况下，客户端所有的操作都被禁止，所以，需要添加一些 `allow` 规则。记住：服务端代码和methods 不受 `allow` 和 `deny` 影响 — 这些规则只应用于当不可信的客户端代码调用 `insert` , `update` , 和 `remove` 时。

例如，假设只有当 `createBy` 字段为当前用户ID时，才允许用户插入新文章，这样用户就不能冒充其他人

```
// In a file loaded on the server (ignored on the client)
Posts.allow({
  insert: function (userId, post) {
    // can only create posts where you are the author
    return post.createdBy === userId;
  },
  remove: function (userId, post) {
    // can only delete your own posts
    return post.createdBy === userId;
  }
  // since there is no update field, all updates
  // are automatically denied
});
```

`allow` 方法接受三个回调函数： `insert` , `remove` , `update` 。三个回调函数的第一个参数是登录用户的 `_id` ,其余的参数如下：

1. `insert(userId, document)`

   `document` 是将要插入到数据库的文档。如果允许插入，则返回 `true` ，否则返回 `false`

2. `update(userId, document, fieldNames, modifier)`

   `document` 是将要被修改的文档。 `fieldNames` 是一个数组，包含了受这次修改影响的一级字段。 `modifier` 是传给 `collection.update` 的第二个参数Mongo Modifier。 如果使用这个回调无法实现正确的校验，那么推荐使用methods。 如果允许修改，则返回 `true` ，否则返回 `false`

3. `remove(userId, document)`

   `document` 是将要从数据库中移除的文档。如果允许移除则返回 `true` ，否则返回 `false`

collection.deny(options)

Server

Override `allow` rules.

Options

insert, update, remove Function

Functions that look at a proposed modification to the database and return true if it should be denied, even if an allow rule says otherwise.

`deny` 方法允许你选择性的重写 `allow` 规则。只要有一个 `allow` 回调函数返回 `true` ，就允许修改，但必须所有 `deny` 规则都返回false，才允许修改。

例如，我们要重写上面定义的 `allow` 规则：排除特定标题的文章：

```
// In a file loaded on the server (ignored on the client)
Posts.deny({
  insert: function (userId, post) {
    // Don't allow posts with a certain title
    return post.title === "First!";
  }
});
```

Accounts

要增加账户功能，用 `meteor add` 添加下面的一个或多个包：

- `accounts-ui` :这个包允许你通过在模板中使用 `{{> loginButtons}}` ，来添加自动生成的登录UI， 用户可以登录。社区中有其它的替代选择，或者你也可以结合使用advanced Accounts methods
- `accounts-password` : 这个包允许用户通过密码登录。添加之后， `loginButtons` 下拉框会自动增加邮箱和密码文本域。
- `accounts-facebook` , `accounts-google` , `accounts-github` , `accounts-twitter` , 以及其它由社区贡献的第三方登录包，让你的用户可以通过第三方网站登录。 它们会自动添加登录按钮到 `loginButtons` 下拉框中。

{{> loginButtons}} Client

在HTML中引入 `loginButtions` 模板，就可以使用Meteor默认的登录UI。使用前，需要先添加 `accounts-ui` 包：

```
$ meteor add accounts-ui
```

Anywhere but publish functionsMeteor.user()

Get the current user record, or `null` if no user is logged in. A reactive data source.

从 `Meteor.users` 集合中获取当前登录用户。等同于 `Meteor.users.findOne(Meteor.userId())` 。

Anywhere but publish functionsMeteor.userId()

Get the current user id, or `null` if no user is logged in. A reactive data source.

Meteor.users

Anywhere

A Mongo.Collection containing user documents.

这个集合包含了所有注册用户，每个用户是一个文档。例如：

```
{
  _id: "bbca5d6a-2156-41c4-89da-0329e8c99a4f",  // Meteor.userId()
  username: "cool_kid_13", // unique name
  emails: [
    // each email address can only belong to one user.
    { address: "cool@example.com", verified: true },
    { address: "another@different.com", verified: false }
  ],
  createdAt: Wed Aug 21 2013 15:16:52 GMT-0700 (PDT),
  profile: {
    // The profile is writable by the user by default.
    name: "Joe Schmoe"
  },
  services: {
    facebook: {
      id: "709050", // facebook id
      accessToken: "AAACCgdX7G2...AbV9AZDZD"
    },
    resume: {
      loginTokens: [
        { token: "97e8c205-c7e4-47c9-9bea-8e2ccc0694cd",
          when: 1349761684048 }
      ]
    }
  }
}
```

一个用户文档可以包含任何你想保存的用户相关的数据。不过，Meteor会特殊对待下面的几个字段：

- `username`：一个唯一的字符串，可以标识用户。
- `emails`：一个对象的数组。对象包含属性 `address` 和 `verified` 。一个邮箱地址只能属于一个用户。 `verified` 是一个布尔值，如果用户已经验证 邮箱地址则为true。
- `createdAt`：用户文档创建时间。
- `profile`：一个对象，默认情况下用户可以用任何数据新建和更新该字段。
- `services`：包含第三方登录服务使用的数据的对象。例如，它的 `reset` 字段包含的token，用于 忘记密码的超链接，它的 `resume` 字段包含的token，用于维持用户登录状态。

和所有的Mongo.Collection一样，在服务端，你可以获取用户集合 的所有文档，但是在客户端只能获取那些服务端发布的文档。

默认情况下，当前用户的 `username` , `emails` ,和 `profile` 会发布到客户端。 可以使用下面的代码发布当前用户的其它字段：

```
// server
Meteor.publish("userData", function () {
  if (this.userId) {
    return Meteor.users.find({_id: this.userId},
                             {fields: {'other': 1, 'things': 1}});
  } else {
    this.ready();
  }
});

// client
Meteor.subscribe("userData");
```

如果安装了autopublish包，那么所有用户的信息都会发布到所有客户端。包括 `username` , `profile` ,以及 `service` 中所有可以公开的字段(例如： `services.facebook.id` , `services.twitter.screenName`)。另外，使用autopublish时，对于当前登录用户会发布更多的信息， 包括access token。这样就可以直接从客户端发起API调用。

默认情况下，用户可以通过 `Accounts.createUser` 声明自己的 `profile` 字段， 也可以通过 `Meteor.users.update` 来修改它。要允许用户修改更多的字段，使用 `Meteor.users.allow` ，要禁止用户对自己的文档做任何修改，使用：

```
Meteor.users.deny({update: function () { return true; }});
```

{{ currentUser }}

Calls Meteor.user(). Use `{{#if currentUser}}` to check whether the user is logged in.

Methods

Methods 是可以从客户端调用的服务端函数。当你想做一些比 `insert` , `update` 或是 `remove` 复杂的事情，或是进行数据验证，而 `allow` 和 `deny` 又无法满足需求时，使用Methods非常合适。

Methods 可以返回值或是抛出错误。

Meteor.methods(methods)

Anywhere

Defines functions that can be invoked over the network by clients.

Arguments

methods Object

Dictionary whose keys are method names and values are functions.

在服务端调用 `Meteor.methods` 定义的函数可以在客户端远程调用。下面是一个method的例子，检查参数，抛出错误：

```
// On the server
Meteor.methods({
  commentOnPost: function (comment, postId) {
    // Check argument types
    check(comment, String);
    check(postId, String);

    if (! this.userId) {
      throw new Meteor.Error("not-logged-in",
        "Must be logged in to post a comment.");
    }

    // ... do stuff ...

    return "something";
  },

  otherMethod: function () {
    // ... do other stuff ...
  }
});
```

`check` 函数用于确保method的参数是期望的类型和结构。

在method定义中， `this` 绑定到一个method调用对象，method调用对象有几个非常有用的属性，包括：用于标识当前登录用户的 `this.userId` 。

你不用把所有的method定义都放到一个 `Meteor.methods` 里面；可以多次调用 `Meteor.methods` ，只要每个method的名字是唯一的。

Latency Compensation

调用服务端的一个method会产生在网络上一个来回的延迟。如果用户由于这个延迟，需要等待一秒才能看到自己的评论显示出来，会让人非常不爽。这就是为什么Meteor有一个叫做*method stubs* 的功能。如果你在客户端定义了一个和服务端同名的method，Meteor会运行它，尝试预测服务端运行的结果。当服务端代码执行完毕后，客户端生成的预测结果会被实际结果替代。

`insert` , `update` , 和 `remove` 的客户端版本，都是以method方式实现，这样在客户端与数据库进行交互的结果就会立即呈现。

Meteor.call(name, [arg1, arg2...], [asyncCallback])

Anywhere

Invokes a method passing any number of arguments.

Arguments

name String

Name of method to invoke

arg1, arg2... EJSON-able Object

Optional method arguments

asyncCallback Function

Optional callback, which is called asynchronously with the error or result after the method is complete. If not provided, the method runs synchronously if possible (see below).

用上面的方法来调用method。

On the client

在客户端调用的Method是异步执行，为了能够获取执行结果，你需要传入一个回调函数。回调函数会收到两个参数 `error` 和 `result` 。除非有异常抛出，否则 `error` 参数为 `null` 。当有异常抛出时， `error` 参数是 `Meteor.Error` 的一个实例，同时 `result` 参数是undefined。

示例：调用 `commentOnPost` method ，传入两个参数 `comment` 和 `postId` ：

```
// Asynchronous call with a callback on the client
Meteor.call('commentOnPost', comment, postId, function (error, result) {
  if (error) {
    // handle error
  } else {
    // examine result
  }
});
```

作为方法调用的一部分，Meteor会跟踪数据库的更新，直到所有的更新都发送到客户端之后才调用客户端回调函数。

On the server

在服务端，不用传入回调函数 — 方法调用会阻塞直到执行完毕，返回结果或是抛出异常，就好像直接调用函数一样：

```
// Synchronous call on the server with no callback
var result = Meteor.call('commentOnPost', comment, postId);
```

new Meteor.Error(error, [reason], [details])

Anywhere

This class represents a symbolic error thrown by a method.

Arguments

error String

A string code uniquely identifying this kind of error. This string should be used by callers of the method to determine the appropriate action to take, instead of attempting to parse the reason or details fields. For example:

```
// on the server, pick a code unique to this error
// the reason field should be a useful debug message
throw new Meteor.Error("logged-out",
  "The user must be logged in to post a comment.");

// on the client
Meteor.call("methodName", function (error) {
  // identify the error
  if (error.error === "logged-out") {
    // show a nice error message
    Session.set("errorMessage", "Please log in to post a comment.");
  }
});
```

For legacy reasons, some built-in Meteor functions such as `check` throw errors with a number in this field.

reason String

Optional. A short human-readable summary of the error, like 'Not Found'.

details String

Optional. Additional information about the error, like a textual stack trace.

如果想在method中返回一个错误，使用抛出异常。Method可以抛出任意类型的异常，但是只有 `Meteor.Error` 类型的错误会发送到客户端。如果method抛出一个其它类型的异常，客户端会收到 `Meteor.Error(500, 'Internal server error')` 。

发布和订阅

Meteor 服务端可以通过 `Meteor.publish` 发布文档集，同时客户端可以通过 `Meteor.subscribe` 订阅这些发布。 任何客户端订阅的文档都可以通过 `find` 方法进行查询使用。

默认情况下，每个新创建的 Meteor 应用包含有 autopublish 包，它会自动为每个客户端发布所有可用的文档。 为了可以更细化的控制不同客户端所接收的数据文档,首先应该在终端移除 autopublish：

```
$ meteor remove autopublish
```

现在，你可以使用 `Meteor.publish` 和 `Meteor.subscribe` 来控制不同的文档从服务端推送到不同的客户端了。

Meteor.publish(name, func)

Server

Publish a record set.

Arguments

name String

Name of the record set. If `null`, the set has no name, and the record set is automatically sent to all connected clients.

func Function

Function called on the server each time a client subscribes. Inside the function, `this` is the publish handler object, described below. If the client passed arguments to `subscribe`, the function is called with the same arguments.

将数据发布到客户端，通过在服务端调用 Meteor.publish，需要传入两个参数：该记录集的名称，以及一个会在每一次客户端订阅该记录集的时候被调用的发布功能函数。

发布功能通过返回在一些 `collection` 调用 `collection.find(query)` 的结果。通过 `query` 来限制发布的文档集：

```
// 发布已登录用户的文章集合
Meteor.publish("posts", function () {
  return Posts.find({ createdBy: this.userId });
});
```

你可以发布来自多个collection的文档，通过返回一个 `collection.find` 结果集合:

```
// 发布一个单独的文章和对应的评论
Meteor.publish("postAndComments", function (postId) {
  // 检查参数
  check(postId, String);

  return [
    Posts.find({ _id: postId }),
    Comments.find({ postId: roomId })
  ];
});
```

在发布功能里， `this.userId` 是当前登录用户的 `_id` , 在某些文档仅对特定用户可见时，可以用其来过滤集合。 如果客户端登录的用户发生改变，发布功能也将自动使用新的 `userId` ,所以新用户无法访问任何只限于之前登录用户的文档。

Meteor.subscribe(name, [arg1, arg2...], [callbacks])

Client

Subscribe to a record set. Returns a handle that provides `stop()` and `ready()` methods.

Arguments

name String

Name of the subscription. Matches the name of the server's `publish()` call.

arg1, arg2... Any

Optional arguments passed to publisher function on server.

callbacks Function or Object

Optional. May include `onStop` and `onReady` callbacks. If there is an error, it is passed as an argument to `onStop` . If a function is passed instead of an object, it is interpreted as an `onReady` callback.

客户端调用 `Meteor.subscribe` 来订阅服务端发布的文档集合。 客户端可以进一步过滤文档集合通过调用 `collection.find(query)` . 当任何通过发布功能发布的可访问的数据在服务端发生改变时，发布功能会自动返回和更新发布的文档集合推送给客户端。

`onReady` 回调函数在服务端已经发送了订阅的所有初始化数据时被调用，无需传入任何参数。 `onStop` 订阅不管以任何原因被终止时调用 ;如果订阅失败是由于服务端错误，它将收到一个 `Meteor.Error` 。

`Meteor.subscribe` 返回一个订阅句柄，是一个包含以下方法的对象：

stop()

取消订阅。这通常会导致服务器引导客户端从客户端缓存中删除订阅的数据。

ready()

如果服务端已经设置状态为ready，将会返回 True。一个反应式数据源。

如果你在 `Tracker.autorun` 里调用 `Meteor.subscribe` , 当运算返回时，订阅将被自动取消(所以如果合适的话，一个新的订阅会被创建), 意味着你不能够在来自 `Tracker.autorun` 里的订阅中调用 `stop` 。

Environment

Meteor.isClient

Anywhere

Boolean variable. True if running in client environment.

Meteor.isServer

Anywhere

Boolean variable. True if running in server environment.

`Meteor.isServer` 可以用来限制代码的运行位置，但是它不会阻止代码发送到客户端。任何你不想发送到客户端的敏感代码，例如包含密码或是认证机制的代码，都应该放到 `server` 文件夹。

Meteor.startup(func)

Anywhere

Run code when a client or a server starts.

Arguments

func Function

A function to run on startup.

在服务端，只要服务进程启动完成，回调函数就会执行。在客户端，只要页面ready，回调函数就会执行。

最佳实践是：把模板事件，模板Helper， `Meteor.methods` , `Meteor.publish` , 或是 `Meteor.subscribe` 之外的代码包裹进 `Meteor.startup` ，这样APP的代码就不会在环境准备好之前运行。

例如：当服务端启动时，如果数据库为空则创建一些初始数据，可以用下面的方式：

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    if (Rooms.find().count() === 0) {
      Rooms.insert({name: "Initial room"});
    }
  });
}
```

如果在服务端进程启动完成之后，或是客户端，页面ready之后调用 `Meteor.startup` ，回调函数会立即执行。

Packages

Meteor所有的功能都是以模块化的包实现的。除了上面提到的核心包之外，还有很多可以用来增加功能的包。

在命令行，添加和删除包使用 `meteor add` 和 `meteor remove`：

```
# add the less package
meteor add less

# remove the less package
meteor remove less
```

当添加或是删除包的时候，APP会自动重启。APP的包依赖通过 `.meteor/packages` 记录跟踪， 当你的同事更新项目源代码后，会自动更新为相同的包，因为你们的 `.meteor/packages` 文件是相同的。

在APP目录下，运行 `meteor list` 可以查看APP使用了哪些包。

Searching for packages

目前查找包最好的方式就是通过官方的Meteor包服务器Atmosphere。也可以直接通过命令 `meteor search` 搜索包。

包名中间带 `:` 的，例如： `mquandalle:jade` ，说明是由社区成员编写和维护的。冒号前边是创建这个包的成员或组织名称。不带冒号的，说明是由Meteor Development Group维护，作为Meteor框架的一部分。

当前在Atmosphere上有超过两千个包。下面列出了一些非常有用的包。

accounts-ui

Meteor 账户系统的插入式UI。添加这个包之后，用 `{{> loginButtons}}` 插入到模板中。UI自动匹配任何已添加的登录服务，例如 `accounts-password`，`accounts-facebook` 等等。

See the docs about accounts-ui above..

coffeescript

在APP中使用CoffeeScript。添加了这个包之后，所有以 `.coffee` 后缀结尾的文件都会由Meteor的构建系统编译为javascript。

email

发送邮件。参见email section of the full API docs

mquandalle:jade

在APP中使用Jade作为模板语言。添加包之后，任何以`.jade`作为后缀的文件都会编译为Meteor模板。详情参见page on Atmosphere

jquery

JQuery让跨浏览器进行HTML遍历，操作，事件处理，和动画操作变得简单。

JQuery自动添加到每个Meteor APP中，因为框架大量的使用了jQuery。详情参见JQuery docs 。

http

利用这个包可以在客户端和服务端使用相同的API发送HTTP请求。使用方法参见http docs。

less

添加LESS CSS预处理器到APP，`.less`后缀的文件都会编译为常规CSS。如果想使用`@import`引入其他文件，同时又不想让Meteor自动编译它们，使用`.import.less`后缀。

markdown

在模板中插入Markdown代码。使用 `{{# markdown}}` Helper很简单：

```
<div class="my-div">
{{#markdown}}
# My heading

Some paragraph text
{{/markdown}}
</div>
```

确保你的markdown缩进正确。

underscore

Underscore 提供了一系列有用的函数，用来操作数组，对象，和函数。每个Meteor APP都包含 `underscore` 包，因为框架大量使用了underscore。

spiderable

这个包可以让APP在服务端进行渲染，允许搜索引擎爬虫抓取页面内容。如果在意SEO的话，那么应该添加这个包。

查看完整API 文档

恭喜你，你已经看完了Meteor基础文档。更多高级的功能和详细的介绍，请看 完整版 API 文档.